Please replace the paragraph beginning at page 11, line number 24, with the following rewritten paragraph:

A virtual volume manager 260 communicates with storage managers and data storage devices connected to the SAN. The virtual volume manager 260 also manages requests between requesters such as servers and data storage devices connected to the SAN. Generally, each data storage device uses a process known as discovery during its boot-up to register with the manager. Each volume manager broadcasts volume attributes periodically or in case of a query. The SAN manager will then register the volume. When a particular server requests a particular volume with specified characteristics, the SAN manager allocates the storage depending on the parameters requested by the server. When all the storage in the SAN is exhausted, the SAN manager can now query other SANs to request the storage. Alternatively, the attributes of a device may be statically configured in a server or in a switch.

Please replace the paragraph beginning at page 15, line number 1, with the following rewritten paragraph:

Referring now to Fig. 7, the delete function 310 is shown in more detail. The delete function 310 is a counterpart of the create function 308. The delete function 310 first checks whether storage space is to be deleted from a local volume (step 361) and if so, the storage space is released back to the local storage volume (step 363). Alternatively, the delete function 310 then checks whether the deletion is to occur on a SAN volume (step 365). If so, space is deallocated on the SAN volume (step 367). Alternatively if space is to be deleted from the LAN/WAN volume (step 369), the delete function 310 deletes the space from the LAN/WAN storage (step 371). If the volume to be deleted is not resident on the LAN/WAN volume, the delete function 310 then checks whether the deletion is to occur from the storage framework volume (step 373). If so, space is deallocated from the storage framework volume (step 375). From steps 363, 367, 371, or 375, the respective local storage manager, SAN storage manager, LAN/WAN storage manager or storage framework manager is notified with the space deallocation (step 379) and then the function 310 exits.

Please replace the paragraph beginning at page 15, line number 20, with the following rewritten paragraph:

Referring now to Fig. 8, the expand function 312 is shown. In response to a request to allocate more space to an existing volume, the expand function 312 first checks whether additional space is available on other local storage devices (step 428) and, if so, the expand function 312 requests storage space from one of the local storage devices and updates the local storage manager (step 430). Alternatively, if space is not available on any of the local storage devices, the expand function 312 then checks whether additional space is available on other SAN devices (step 432). If so, space is requested from the SAN volume and the appropriate SAN manager is updated (step 434).

A5

Please replace the paragraph beginning at page 15, line number 31, with the following rewritten paragraph:

From step 432, in the event that space is not available on either the local storage device or the SAN device, the expand function 312 then checks whether additional space is available on one of the LAN/WAN storage devices (step 436), and if so, appropriate space is allocated from the LAN/WAN devices (step 438). In this case, the LAN/WAN manager is also updated in step 438. From step 436, if space is not available on the LAN/WAN device, the expand function 312 then checks whether additional space may be available on one of the storage framework devices (step 440). If not, the expand function 312 aborts (step 441). Alternatively, space is allocated on the storage framework device and the storage framework manager is updated (step 442) before the expand function 312 exits.

A6

Please replace the paragraph beginning at page 19, line number 11, with the following rewritten paragraph:

Referring now to Fig. 13, the corresponding detach function 324 is shown. The detach function is the inverse of the attach function and checks whether the device which has been requested to be detached is a local storage device (step 451) or whether the device resides on the SAN, LAN/WAN, or the storage framework (steps 455, 459, and 463). If so, the detach function

A7 cont.

A7 cmdd.

324 removes the volume from the respective manager and removes the information from the volume table (steps 453, 457, 461 and 465).

Please replace the paragraph beginning at page 19, linen umber 19, with the following rewritten paragraph:

Referring now to Fig. 14, the attribute function 326 is illustrated. The attribute function 326 sequentially checks whether the device whose attribute is being requested is a local storage device (step 650), a SAN device (step 654), a LAN/WAN device (step 658), or a storage framework device (step 662). If so, it queries and returns the volume attributes from the local storage manager (step 652), the SAN manager (step 656), the LAN/WAN manager (step 660) or from the framework manager (step 664), respectively. Similarly, the initialization function 332 sequentially goes through and initializes each data storage device that resides on the local storage device, the SAN, the LAN/WAN or the storage framework.

A8

Please replace the paragraph beginning at page 21, line number 1, with the following rewritten paragraph:

In this example, NFS generates a file system write which calls a block I/O (bio) write which calls the virtual partition strategy module, which calls the RAID strategy module 478, which in turn calls the disk strategy module 480 that actually writes the data to the disk. The virtual partition module can either stripe the data across all the members or can mirror data.

A9

Please replace the paragraph beginning at page 21, line number 24 and ending at page 23, line number 9, with the following rewritten paragraph:

The RAID strategy module 478 is a part of a typical device switch structure

A10 Cmt

```
struct bdevsw {
    int    (*d_open)(dev_t *, int, int, struct cred *);
    int    (*d_close)(dev_t, int, int, struct cred *);
    void   (*d_strategy)(struct buf *);
    int    (*d_ioctl)(dev_t, int, int, int, struct cred *, int *);
```

```
    int    (*d_lookup_path)(char *, dev_t *);
    int    (*d_size)(dev_t);
    char   *d_name;
    void   *d_priv;

};

Enum VP_ATTR {
    VP_INIT,      /* initialize the virtual partition */
    VP_ATTATCH,   /* attach the virtual partition */
    VP_DETACH,    /* detach the virtual partition */
    VP_CONNECT,   /* connect the virtual partition */
    VP_BIND,      /* bind to the storage */
    VP_WRITE,     /* write data */
    VP_READ,      /* read data */
    VP_MOVE,      /* move data */
    VP_SIGNAL,        /* signal events */
    VP_QUERY,     /* query the virtual partition, can also run discovery */
    VP_CONVERT,   /* data format conversion */
    VP_CONTROL,   /* control point */
    VP_CALLBACK,  /* call back function */
    VP_MISC1,     /* miscellaneous, user defined */
    VP_MISC2,
    VP_MISC3,
    VP_MISC4
};
```

Additionally, a structure vpops{} is defined to support the above attributes as follows:

```
Struct vpops {
    Vp_init(),
```

```
        Vp_attach(),
        Vp_detach(),
        Vp_connect(),
        Vp_bind(),
        Vp_write(),
        Vp_read(),
    Vp_move(),
    Vp_signal(),

        Vp_query(),
        Vp_convert(),
        Vp_control(),
        Vp_callback(),
        Vp_misc1(),
        Vp_misc2(),
        Vp_misc3(),
        Vp_misc4()
    };
```


Please replace the paragraph beginning at page 25, line number 15, with the following rewritten paragraph:

To illustrate, an exemplary configuration shown in Fig. 8 for video streaming applications will be discussed next. In this video streaming example, a video data provider might choose to play advertisements periodically which cannot be fast-forwarded by a user. The advertisement can be a pointer that stores the actual location of the data. This location can be another member of the virtual partition. When the user fast-forwards the video and tries to read beyond the advertisement, the vp_strategy module can detect this bypass attempt and route the advertisement to the user.